Betriebssysteme (BS)

VL 12.1 – Gerätetreiber – Anforderungen

Volkmar Sieh / Daniel Lohmann

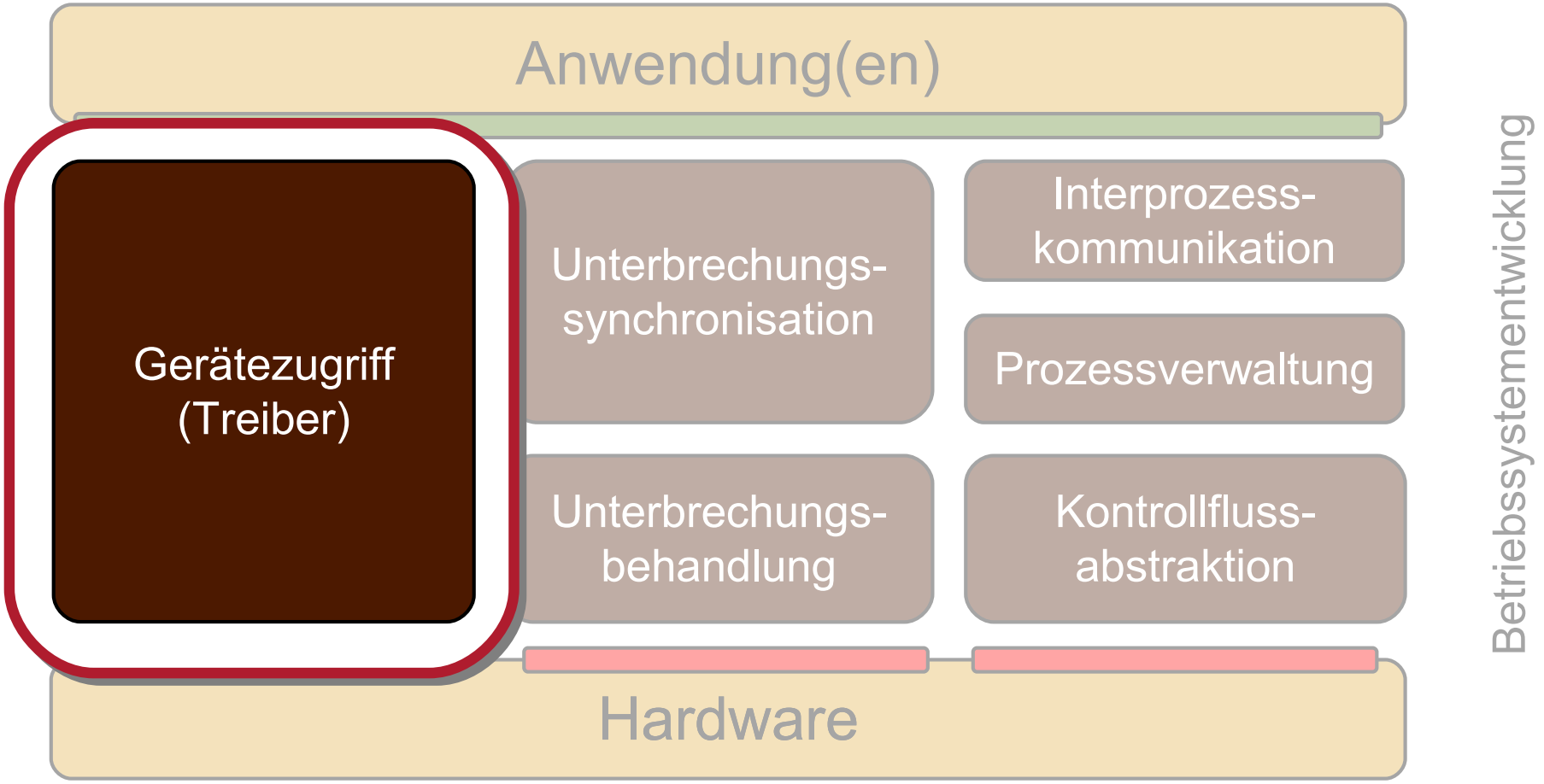
Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen Nürnberg

WS 20 – 1. Februar 2021

https://www4.cs.fau.de/Lehre/WS20/V_BS

Überblick: Einordnung dieser VL



Agenda

Einordnung
Anforderungen an das BS
Struktur des E/A-Systems
Zusammenfassung

Agenda

- Einordnung
  - Bedeutung von Gerätetreibern
- Anforderungen an das BS
- Struktur des E/A-Systems
- Zusammenfassung

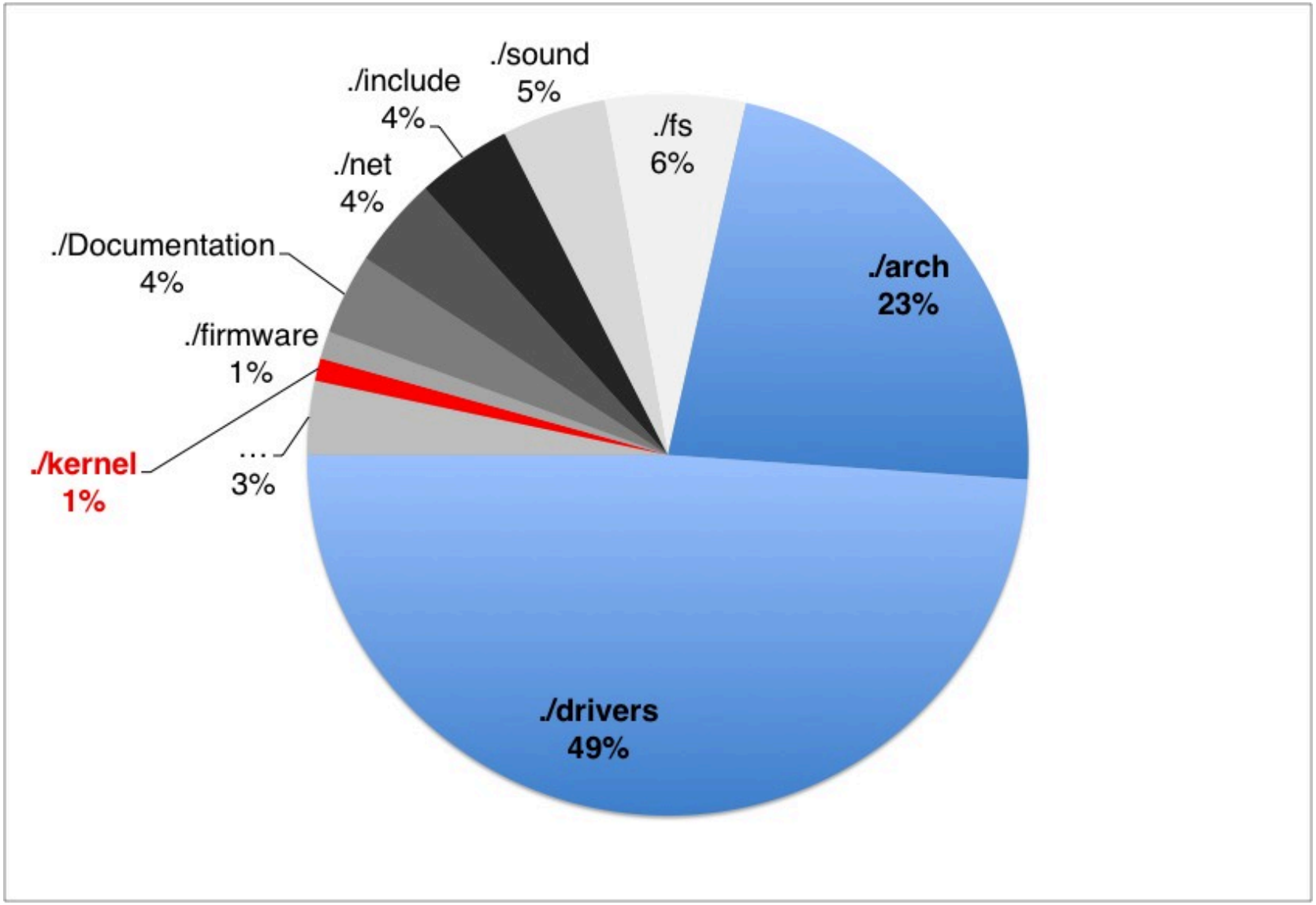
Bedeutung von Gerätetreibern (1)

Anteil Treiber-Sourcen in Linux-5.10.3:

```
linux-5.10.3> du -skh * | sort -n
...
  4.2M  mm
  6.2M  lib
 11M    kernel
 34M    net
 41M    sound
 43M    fs
 43M    tools
 47M    include
 53M    Documentation
134M    arch
670M    drivers
```

Bedeutung von Gerätetreibern (1)

- Anteil an Treibercode in Linux 3.2.1



Bedeutung von Gerätetreibern (2)

- in Linux (3.2.1) ist der Treibercode (ohne ./arch) etwa **50 mal so groß** wie der Code des Kernels
  - und wächst rasant!
    - bei V2.6.32 waren es "nur" 25 mal mehr
    - bei V2.6.11 waren es "nur" 10 mal mehr
  - Windows unterstützt noch deutlich mehr Geräte ...
- Treiberunterstützung ist für die Akzeptanz eines Betriebssystems ein **entscheidender Faktor**
  - warum sonst wäre Linux weiter verbreitet als andere freie UNIXe?
- in Gerätetreibern steckt eine erhebliche Arbeitsleistung

➔ der Entwurf des E/A Subsystems erfordert viel Geschick

- möglichst viele wiederverwendbare Funktionen in eine **Treiber-Infrastruktur** verlagern
- klare Vorgaben bzgl. Treiberstruktur, -verhalten und -schnittstellen, d.h. ein **Treibermodell**

Agenda

- Einordnung
- Anforderungen an das BS
  - Einheitlicher Zugriff
  - Spezifischer Zugriff
- Struktur des E/A-Systems
- Zusammenfassung

Anforderungen an Betriebssysteme

- Ressourcenschonender Umgang mit Geräten
  - schnell arbeiten
  - Energie sparen
  - Speicher, *Ports* und *Interrupt*-Vektoren sparen
  - Aktivierung und Deaktivierung zur Laufzeit
  - Generische *Power Management* Schnittstelle
- Einheitlicher Zugriffsmechanismus
  - **minimaler Satz von Operationen** für verschiedene Gerätetypen
  - **mächtige Operationen** für vielfältige Typen von Anwendungen
- auch gerätespezifische Zugriffsfunktionen

Linux – einheitlicher Zugriff (1)

```
echo "Hallo, Welt" > /dev/ttyS0
```

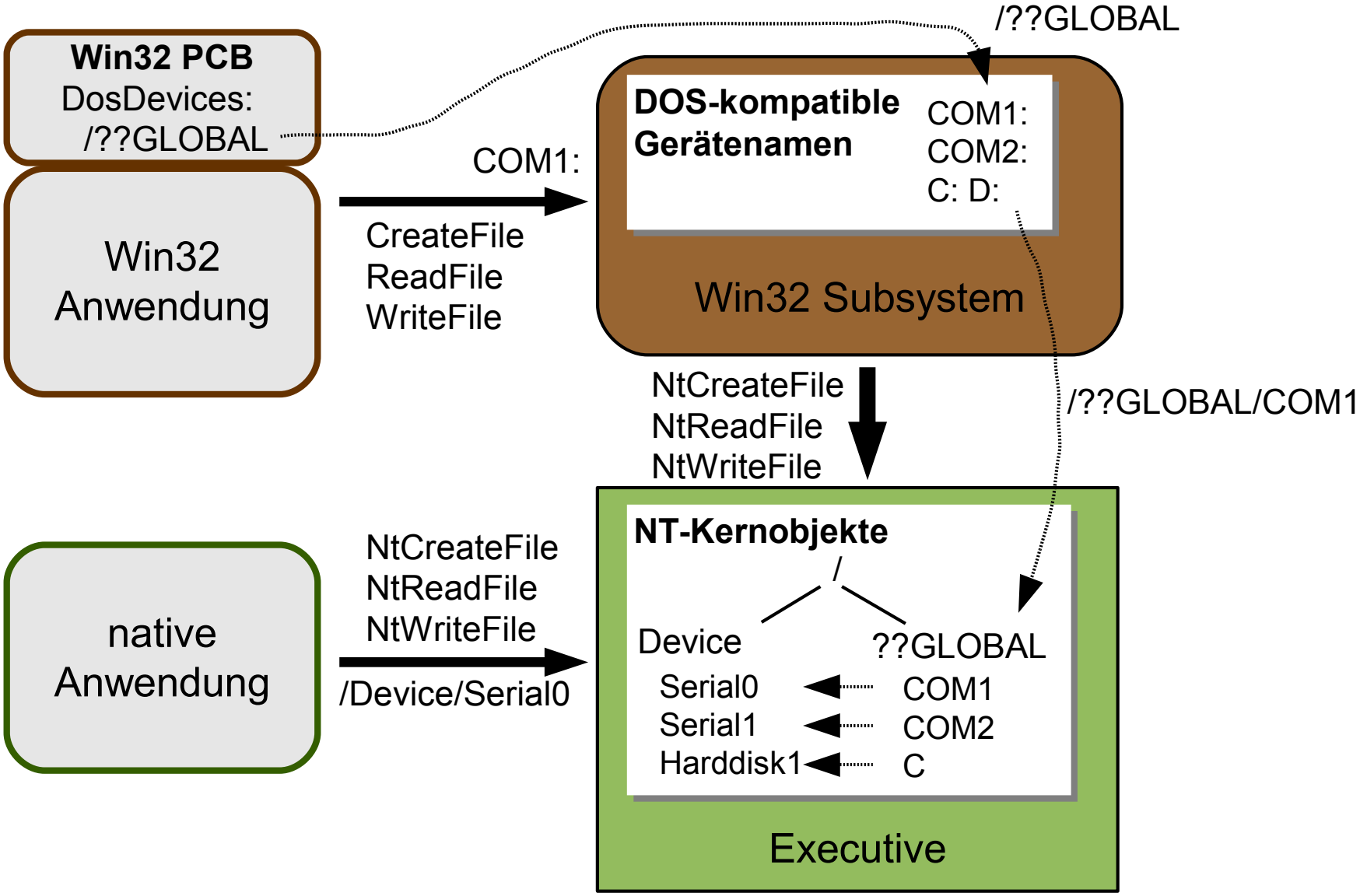
- Geräte sind über Namen im Dateisystem ansprechbar
- Vorteile:
  - Systemaufrufe für Dateizugriff (`open`, `read`, `write`, `close`) können auch für sonstige E/A verwendet werden
  - Zugriffsrechte können über die Mechanismen des Dateisystems gesteuert werden
  - Anwendungen sehen keinen Unterschied zwischen Dateien und "Gerätedateien"
- Probleme:
  - blockorientierte Geräte müssen in Byte-Strom verwandelt werden
  - manche Geräte lassen sich nur schwer in dieses Schema pressen
    - Beispiel: 3D Graphikkarte

Linux – einheitlicher Zugriff (2)

- blockierende Ein-/Ausgabe (Normalfall)
  - `read`: Prozess blockiert bis die angeforderten Daten da sind
  - `write`: Prozess blockiert bis Schreiben möglich ist
- nicht-blockierende Ein-/Ausgabe
  - `open`/`read`/`write` mit dem Zusatz-Flag `O_NONBLOCK`
  - statt zu blockieren kehren `read` und `write` so mit `-EAGAIN` zurück
  - der Aufrufer kann/muss die Operation später wiederholen
- nebenläufige Ein-/Ausgabe
  - neu: `aio_(read|write|...)` (POSIX 1003.1-2003)
  - indirekt mittels Kindprozess (`fork`/`join`)
  - `select`, `poll` Systemaufrufe

Windows – einheitlicher Zugriff (1)

- Geräte sind Kern-Objekte der Executive



Windows – einheitlicher Zugriff (2)

- synchrone oder asynchrone Ein-/Ausgabe

```
BOOL ReadFile(
  HANDLE hFile,
  LPVOID lpBuffer,
  DWORD nNumberOfBytesToRead,
  LPDWORD lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

NULL: synchrones Lesen

```
BOOL GetOverlappedResult(
  HANDLE hFile,
  LPOVERLAPPED lpOverlapped,
  LPDWORD lpNumberOfBytesTransferred,
  BOOL bWait
);
```

true: auf Ende warten
false: Status erfragen

- weitere Möglichkeiten:
  - E/A mit *Timeout*
  - WaitForMultipleObjects – warten auf 1–N Kernobjekte
    - Datei-Handles, Semaphore, Mutex, Thread-Handle, ...
  - I/O Completion Ports
    - Aktivierung eines wartenden Threads nach I/O Operation

Linux – gerätespez. Funktionen (1)

- spezielle Geräteeigenschaften werden (klassisch) über **ioctl** angesprochen:

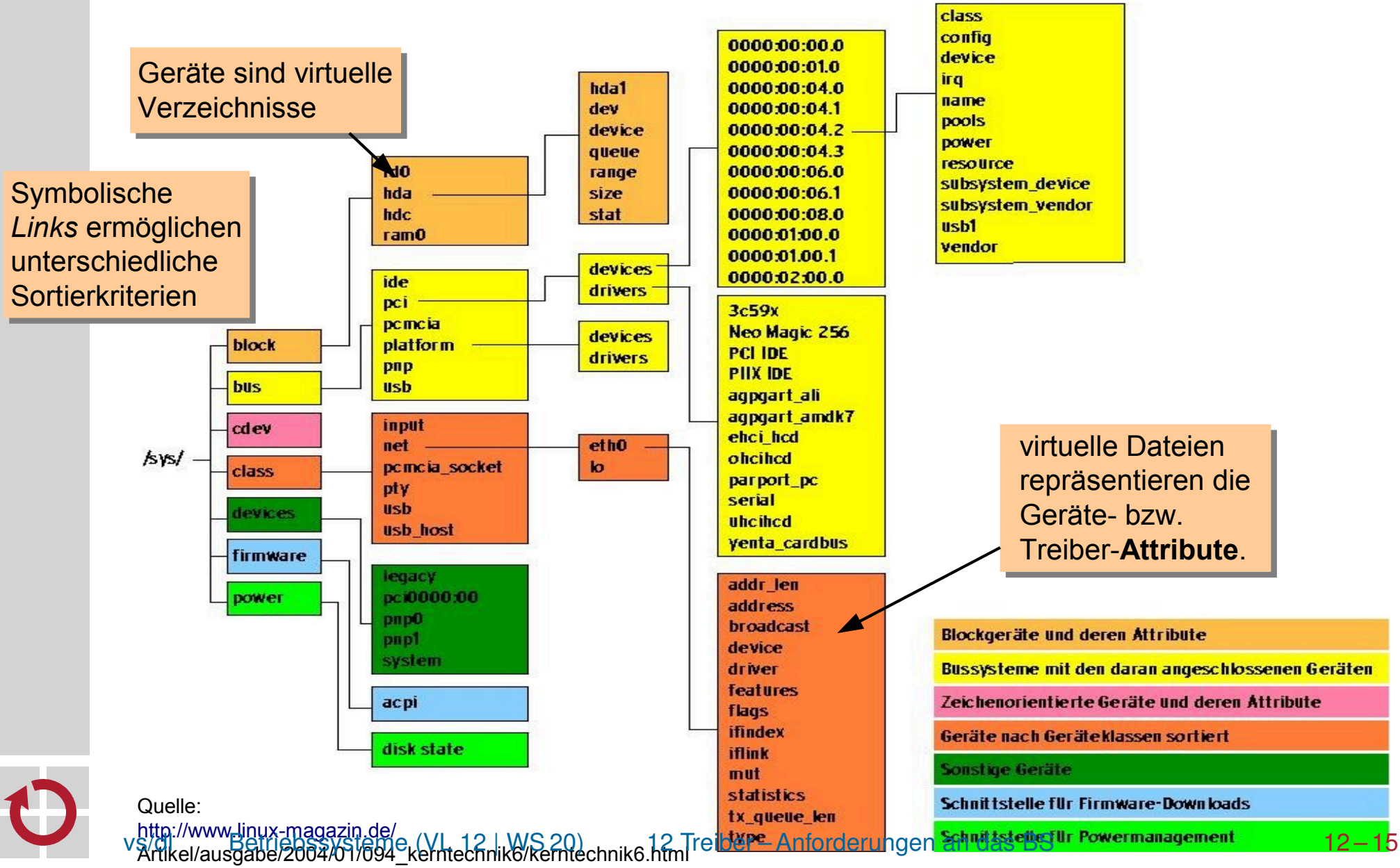
```
IOCTL(2)          Linux Programmer's Manual          IOCTL(2)

NAME
       ioctl - control device

SYNOPSIS
       #include <sys/ioctl.h>

       int ioctl(int d, int request, ...);
```

- Schnittstelle generisch und Semantik gerätespezifisch:

```
CONFORMING TO
       No single standard. Arguments, returns, and semantics of
       ioctl(2) vary according to the device driver in question
       (the call is used as a catch-all for operations that
       don't cleanly fit the Unix stream I/O model). The ioctl
       function call appeared in Version 7 AT&T Unix.
```

Linux – gerätespez. Funktionen (2)

Linux 2.6 – das **Gerätemodell im sys-Dateisystem**



Quelle:
http://www.linux-magazin.de/Artikel/ausgabe/2004/01/094_kerntechnik6/kerntechnik6.html

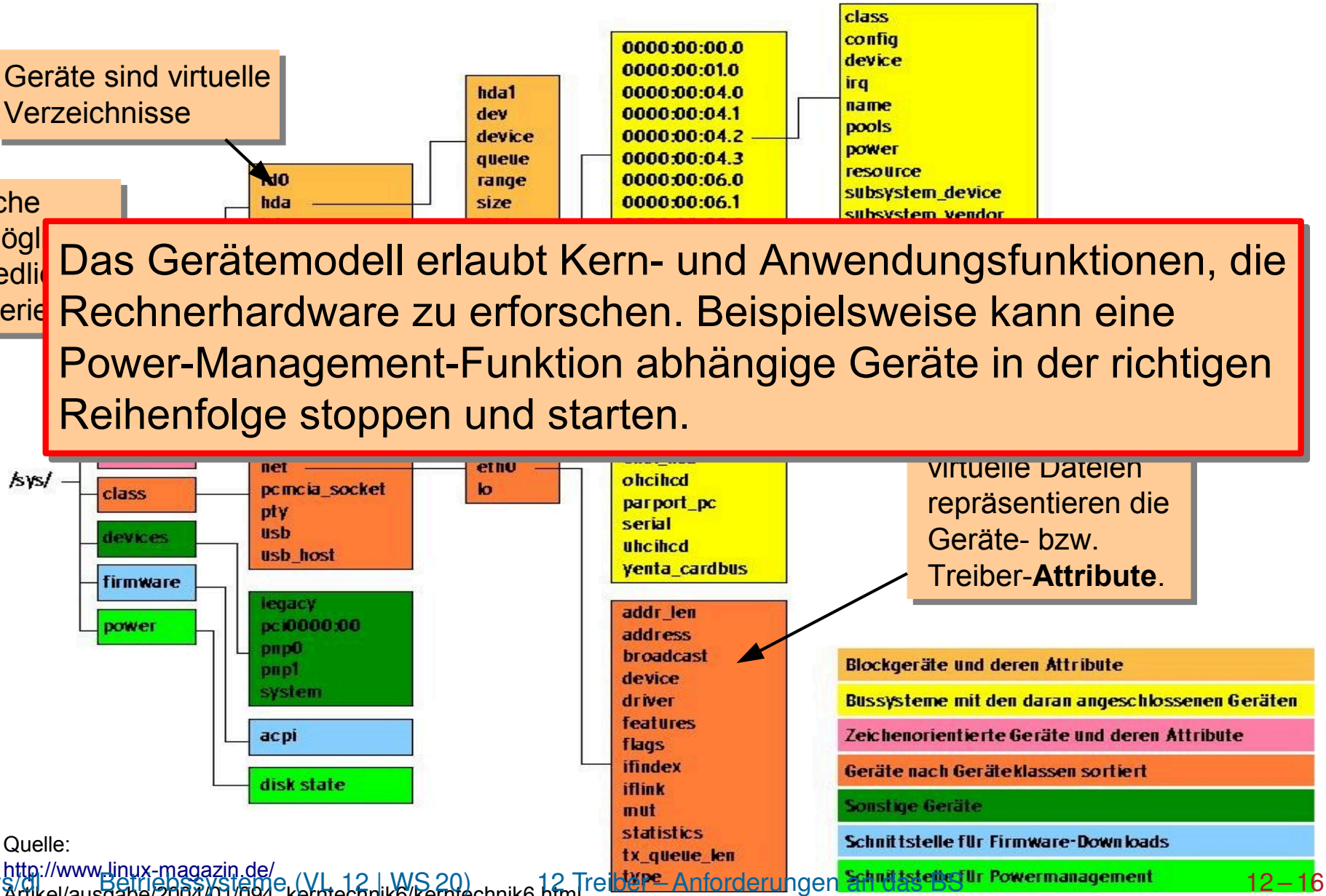
Linux – gerätespez. Funktionen (2)

Linux 2.6 – das **Gerätemodell im sys-Dateisystem**

Geräte sind virtuelle Verzeichnisse

Symbolische *Links* ermögl
unterschiedli
Sortierkriterie

Das Gerätemodell erlaubt Kern- und Anwendungsfunktionen, die Rechnerhardware zu erforschen. Beispielsweise kann eine Power-Management-Funktion abhängige Geräte in der richtigen Reihenfolge stoppen und starten.



Quelle:
http://www.linux-magazin.de/Artikel/ausgabe/2004/01/094_kerntechnik6/kerntechnik6.html

Windows – gerätespez. Funktionen

- **DeviceIoControl** entspricht dem UNIX ioctl:

```
BOOL DeviceIoControl(
  HANDLE hDevice,
  DWORD dwIoControlCode,
  LPVOID lpInBuffer,
  DWORD nInBufferSize,
  LPVOID lpOutBuffer,
  DWORD nOutBufferSize,
  LPDWORD lpBytesReturned,
  LPOVERLAPPED lpOverlapped
);
```

Kommunikation über untypisierte Puffer direkt mit dem Treiber

auch asynchron möglich

- und was sonst?
  - alle Geräte und Treiber werden durch Kern-Objekte repräsentiert
    - spezielle Systemaufrufe gestatten das Erforschen dieses Namensraums
  - statische Konfigurierung erfolgt über die Registry
  - dynamische Konfigurierung erfolgt z.B. über WMI
    - *Windows Management Instrumentation*